Experiment No.8
Implementation of Views and Triggers
Date of Performance:
Date of Submission:

Aim :- Write a SQL query to implement views and triggers

Objective :- To learn about virtual tables in the database and also PLSQL constructs

Theory:

SQL Views:

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

SQL CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;

SQL Dropping a View
A view is deleted with the DROP VIEW statement.
SQL DROP VIEW Syntax

DROP VIEW view_name;

Trigger: A trigger is a stored procedure in the database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger

[trigger_name]

[before | after]

{insert |

update |

delete} on

[table_nam

e]

[for each

row]

[trigger_bo

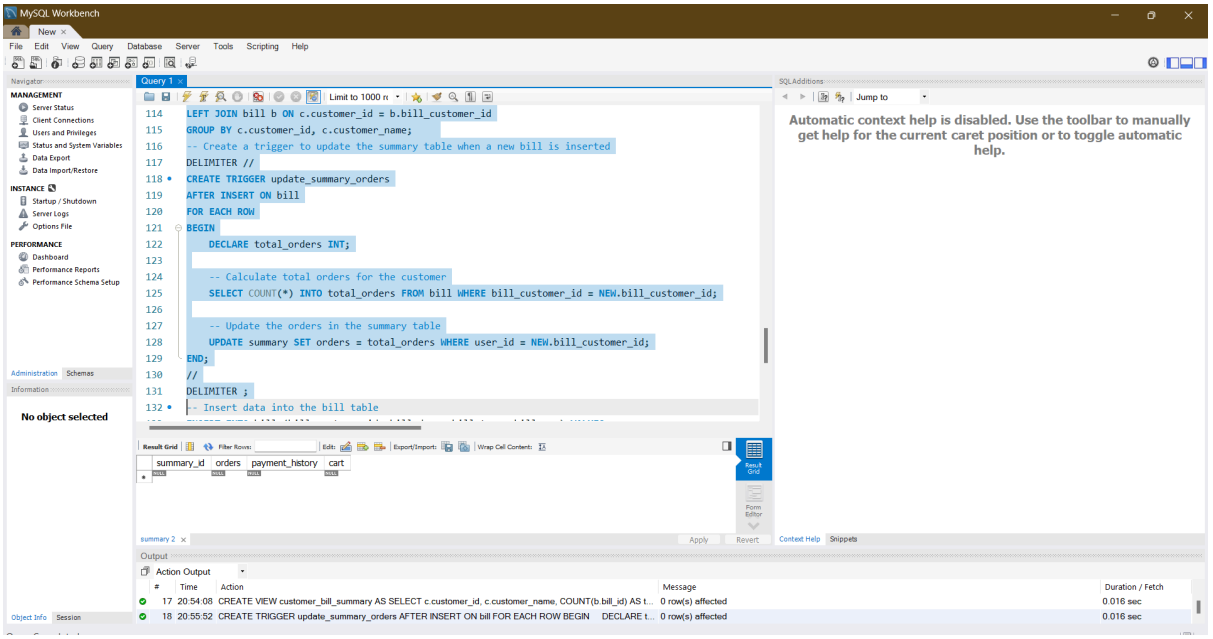
dy]

Explanatio

n of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

Implementation:



Conclusion:

1. Brief about the benefits for using views and triggers.
2. Explain different strategies to update views

In conclusion, using views and triggers in database management offers several benefits:

1. Benefits of Views:
   - Simplified Data Access: Views provide a simplified interface to access complex data by abstracting away the underlying table structure and joining multiple tables into a single virtual table.
   - Data Security: Views can restrict access to specific columns or rows of data, ensuring that users only see the data they are authorized to access.
   - Performance Optimization: Views can optimize query performance by pre-computing and storing frequently accessed data or aggregations, reducing the need for complex queries.
   - Data Consistency: Views can enforce data consistency by presenting a consistent and uniform representation of data across different applications and users.

2. Benefits of Triggers:
   - Automation: Triggers automate repetitive tasks and enforce business rules by automatically executing SQL statements in response to specified events, such as INSERT, UPDATE, or DELETE operations.
   - Data Integrity: Triggers maintain data integrity by enforcing referential integrity constraints, auditing changes, and preventing invalid data modifications.
   - Complex Logic: Triggers allow complex business logic and validation rules to be encapsulated within the database, ensuring consistent enforcement across all database operations.
   - Logging and Auditing: Triggers can be used to log and audit changes to database tables, providing an audit trail for compliance and troubleshooting purposes.

Regarding strategies to update views, several approaches can be employed:

1. Simple Update Strategy:

- Update the underlying base tables directly, and the changes will be reflected in the view automatically.

2. Force View Recomputation:
- Recompute the view by dropping and recreating it whenever the underlying data changes. This can be done manually or through scheduled jobs.

3. Materialized Views:
- Use materialized views, which store the results of a query physically and update them periodically based on a defined schedule or upon changes to the underlying data.

4. Instead of Triggers:
- Use instead of triggers to intercept DML operations on the view and propagate changes to the underlying base tables accordingly.

5. Triggers on Base Tables:
- Implement triggers on the base tables to automatically update the view whenever the underlying data changes.

By employing these strategies, views can be effectively updated to reflect changes in the underlying data, ensuring that users always have access to the most up-to-date information.